LTL: BMC and passive learning

XU Thomas

Adrien Pommellet, LRE

January 14, 2025

Introduction

Motivations

Check that certain properties are verified by our program.

Reactive systems[1]

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Temporal properties to check[1]

Properties to check

For reactive systems, correctness depends on the executions of the system.

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Kripke structure[1]

Definition

A kripke system is a structure M $= \langle Q, I, AP, R \rangle$ where:

- Q: States of the kripke.
- I: Initial states of the kripke.
- AP: Atomic propositions.
- R: QxQ the transition function.

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

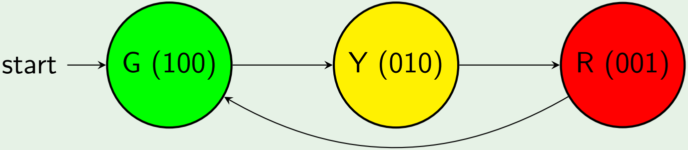
Kripke structure[1]

Definition

A kripke system is a structure $M = \langle Q, I, AP, R \rangle$ where:

- Q: States of the kripke.
- I: Initial states of the kripke.
- AP: Atomic propositions.
- R: QxQ the transition function.

Example: Traffic light modelization



[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Linear temporal logic (LTL)[1]

Problem

Some properties are very hard / impossible to verify by manual testing.

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Linear temporal logic (LTL)[1]

Problem

Some properties are very hard / impossible to verify by manual testing.

LTL formula

1. Atomic propositions (ie.r, g, y)
2. Boolean connectors (and or)
3. Basic temporal operators + Until and Next

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

LTL semantics

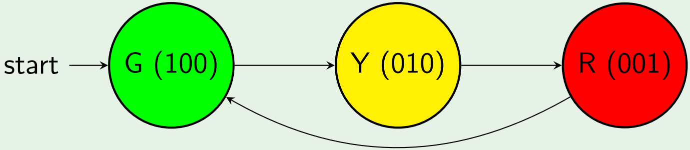
Definition

For an infinite path π of a Kripke structure M and a LTL formula f, we define that f holds on π written $\pi \models$ f:

- $\pi \models p$ iff $p \in L(\pi(0))$.
- $\pi \models Xf$ iff $\pi_1 \models f$.
- $\pi \models Gf$ iff $\pi_i \models f \; \forall i \geq 0$.
- $\pi \models Ff$ iff $\pi_i \models f$ for some i $\geq$ 0.
- $\pi \models fUg$ iff $\pi_i \models g$ for some i $\geq$ 0 and $\pi_j \models f \; \forall 0 \leq j < i$.
- $\pi \models fRg$ iff $\pi_i \models g$ if $\forall j < i$, $\pi_j \not\models f$.

Model Checking Example[1]

Kripke T



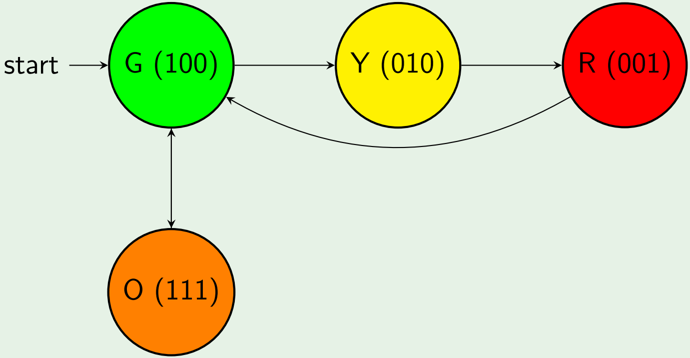
LTL formula

For instance $T \models 100U010$ or $T \not\models G\ 010$

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Model Checking Example[1]

Kripke T



LTL formula

For instance $T \models GF100$ or $T \not\models G\neg 111$

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Model Checking[1]

Pros

Fully automated and returns a counter-example when there is a problem.

Cons

Scales badly with the size of the system.

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.

Bounded Model Checking[1][2][3]

Note

LTL formulas are defined over all paths $\implies$ Finding a counterexample is equivalent to finding a trace that contradicts it.

[1]Christel Baier and Joost-Pieter Katoen: Principles of model checking.
[2]Tzu-Han Hsu et al: Bounded Model Checking for Asynchronous Hyperproperties.
[3]Armin Biere et al: Bounded Model Checking

Bounded Model Checking[1][2][3]

Note

LTL formulas are defined over all paths $\implies$ Finding a counterexample is equivalent to finding a trace that contradicts it.

General idea

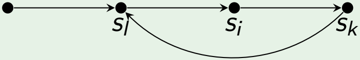
We will try to find counterexamples of size k bounded by considering finite prefix of paths that may be a witness.

[1] Christel Baier and Joost-Pieter Katoen: Principles of model checking.
[2] Tzu-Han Hsu et al: Bounded Model Checking for Asynchronous Hyperproperties.
[3] Armin Biere et al: Bounded Model Checking

Bounded path

No loop

k-l loop



Definition (k-l)-loop

A path π is a (k,l)-loop if
-for $l \leq k$, T(π(k), π(l))
-$\pi = uv^w$ with u = (π(0), ..., π(l-1)) and v = (π(l), ..., π(k)).

Bounded semantics

Definition

Let $k \geq 0$, an LTL formula f is valid along the path π with bound k (written $\pi \models_k$ f) iff:

- π is a k-loop and $\pi \models f$.
- π is not a k-loop and $\pi \models_k^0$ f where:
  - $\pi \models_k^i Xf$ iff $i < k$ and $\pi \models_k^{i+1} f$.
  - $\pi \models_k^i Gf$ is false.
  - $\pi \models_k^i Ff$ iff $\exists j,\ i \leq j \leq k,\ \pi \models_k^j f$.
  - $\pi \models_k^i fUg$ iff $\exists j,\ i \leq j \leq k,\ \pi \models_k^j g$ and $\forall n, i \leq n < j, \pi \models_k^n g$.

Lemmas

Let f be an LTL formula, M a Kripke structure and π a path.

$\pi \models_k f \implies \pi \models f$.

$M \models f \implies \exists k \geq 0$ such that $M \models_k f$.

BMC to SAT

Propositional formula

Given a Kripke structure M, an LTL formula f and a bound k, we will construct a propositional formula $[\![M, f]\!]_k$. Let $s_0, ..., s_k$ be a finite sequence of states on path π.
$[\![M, f]\!]_k$ encodes $s_0, ..., s_k$ such that $[\![M, f]\!]_k$ is satisfiable iff π is a witness for f.

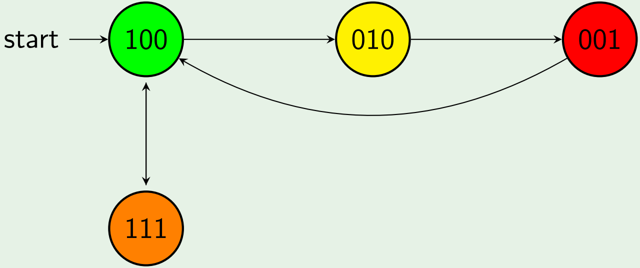
Propositional formula $[\![M, f]\!]_k$

Transition relation

$$[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}).$$

Propositional formula $[\![M, f]\!]_k$

Kripke T



Transition relation

$I(s_0) = s[0] \wedge \neg s[1] \wedge \neg s[2] = s[0]$
$T(s, s') = (s[0] \wedge ((\neg s[1] \wedge \neg s[2]) \leftrightarrow (s'[1] \wedge s'[2]))) \vee (\neg s[0] \wedge s'[1])...$
$[\![M]\!]_2 = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2).$

Propositional formula $[\![M, f]\!]_k$

Loop condition

For a path π, ${}_lL_k$ is true if $T(s_k, s_l)$.
The loop condition L_k is true iff there is a back loop from state k to a previous state or itself:

$$L_k = \bigvee_{l=0}^{k} {}_lL_k$$

Translation of LTL formula

Let f be an LTL formula, k,l,i ≥ 0, with $l, i \leq k$.

Translation for loops

$$_l[\![Gf]\!]_k^i =_l [\![f]\!]_k^i \wedge_l [\![Gf]\!]_k^{succ(i)}$$
$$_l[\![Ff]\!]_k^i =_l [\![f]\!]_k^i \vee_l [\![Ff]\!]_k^{succ(i)}$$
$$_l[\![Xf]\!]_k^i =_l [\![f]\!]_k^{succ(i)}$$
$$_l[\![fUg]\!]_k^i =_l [\![g]\!]_k^i \vee ({}_l[\![f]\!]_k^{succ(i)} \wedge_l [\![fUg]\!]_k^{succ(i)})$$

Translation without loops

$$_l[\![Gf]\!]_k^i =_l [\![f]\!]_k^i \wedge_l [\![Gf]\!]_k^{i+1}$$
...
$$_l[\![f]\!]_k^{k+1} = false$$

Translation of LTL formula

General translation

$$[\![M, f]\!]_k = [\![M]\!]_k \wedge ((\neg L_k \wedge [\![f]\!]_k^l) \vee \bigvee_{l=0}^{k} ({}_l L_k \wedge_l [\![f]\!]_k^0))$$

$[\![M, f]\!]_k$ is satisfiable iff $M \models_k f$.

Propositional formula

Example with T 1/2

The safety property can be G$\neg p$ where $p = s[0] \wedge s[1] \wedge s[2]$. For BMC we want to look for a witness for Fp.

With k = 2, we have for paths without loops:

$[\![Fp]\!]_2^0 = p(s_0) \vee [\![Fp]\!]_2^1$
$[\![Fp]\!]_2^1 = p(s_1) \vee [\![Fp]\!]_2^2$
$[\![Fp]\!]_2^2 = p(s_2) \vee [\![Fp]\!]_2^3$
$[\![Fp]\!]_2^3 = 0$
$[\![Fp]\!]_2^0 = p(s_0) \vee p(s_1) \vee p(s_2)$

Propositional formula

Example with T 2/2

$$[\![M, Fp]\!]_2 = [\![M]\!]_2 \wedge \left((\neg L_k \wedge [\![Fp]\!]_2^l) \vee \bigvee_{l=0}^{2} ({}_lL_2 \wedge_l [\![Fp]\!]_2^0)\right)$$

$$[\![M]\!]_2 = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2)$$

$$[\![Fp]\!]_2^0 = p(s_0) \wedge p(s_1) \wedge p(s_2)$$

$$[\![M, Fp]\!]_2 = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge p(s_0) \wedge p(s_1) \wedge p(s_2)$$

The path $100, 111, 100$ satisfies $[\![M, Fp]\!]_2$. This assignment corresponds to a path from the initial state that violates the safety property.

Passive learning LTL[4][5]

Definition

We have 2 samples of kripke structures P and N, and we want to learn a short LTL formula that distinguish them.

[3] Daniel Neider and Ivan Gavran: Learning Linear Temporal Properties.

[4] Adrien Pommellet et al: SAT-based Learning of Computation Tree Logic.

Work

On going

- Bounded model checking.

Work to do

- Finish it.

Conclusion

Bibliography

Christel Baier and Joost-Pieter Katoen.
Principles of model checking.
MIT Press, 2008.

Tzu-Han Hsu, Borzoo Bonakdarpour, Bernd Finkbeiner, and CÃ©sar SÃ¡nchez.
Bounded model checking for asynchronous hyperproperties, 2023.

Armin Biere et al.
Bounded model checking.
Adv. Comput., 58:117–148, 2003.

Daniel Neider and Ivan Gavran.
Learning linear temporal properties, 2018.

Adrien Pommellet, Daniel Stan, and Simon Scatton.
Sat-based learning of computation tree logic, 2024.